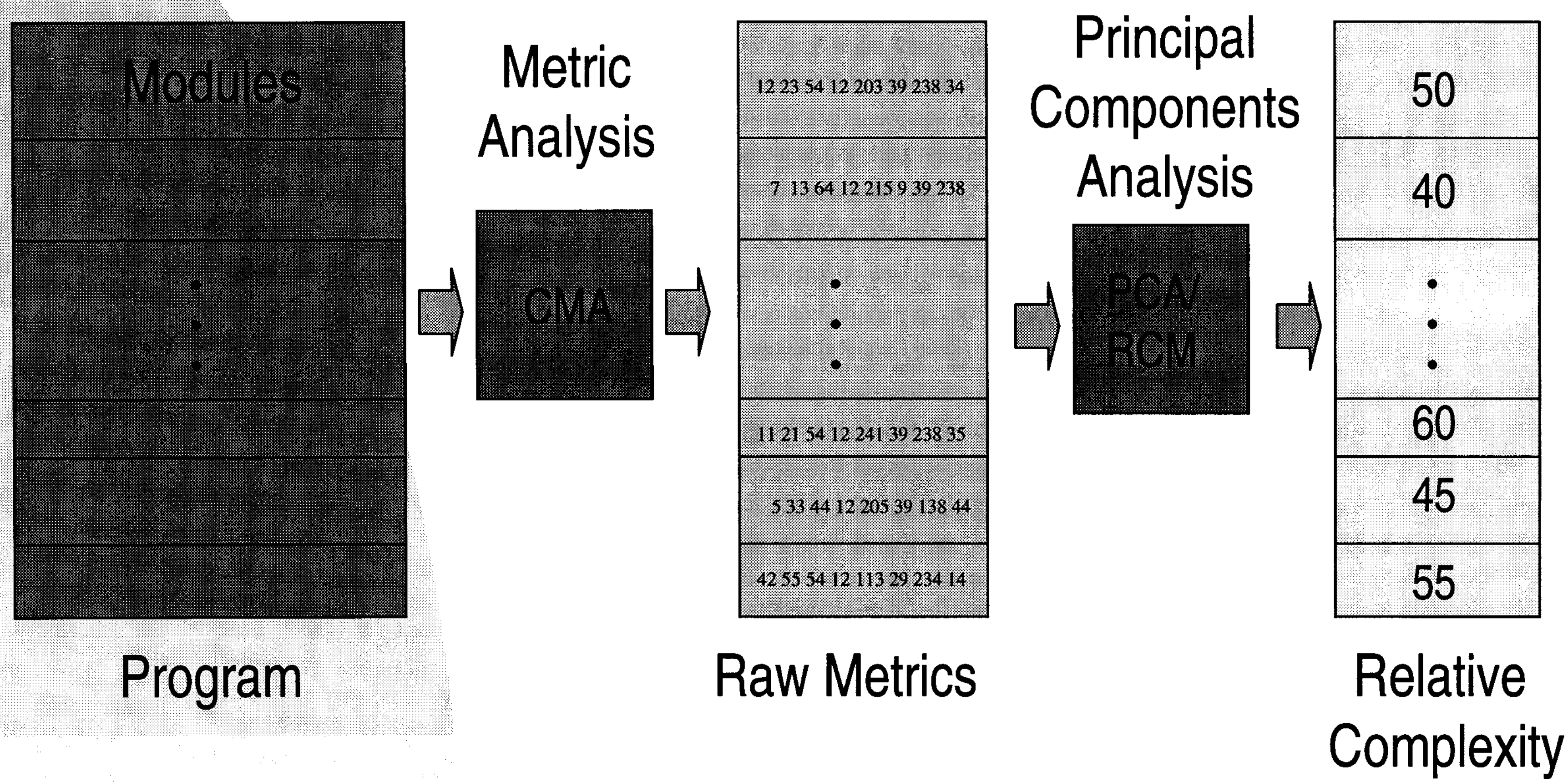
Practical Issues in Measuring Software Quality

Allen P. Nikora
Jet Propulsion Laboratory
California Institute of Technology
Pasadena, CA 91109-8099
Allen.P.Nikora@jpl.nasa.gov

Motivation

- Over the past several years, techniques have been developed to:
  - Estimate a software component's proportional fault burden
  - Use measures of a software component's structural evolution to estimate fault insertion rates
  - Estimate test effectiveness
  - Use risk factors to estimate impacts of a change to a system's quality
- Practical issues:
  - Measuring software structural change
  - Fault identification
  - Obtaining profile information

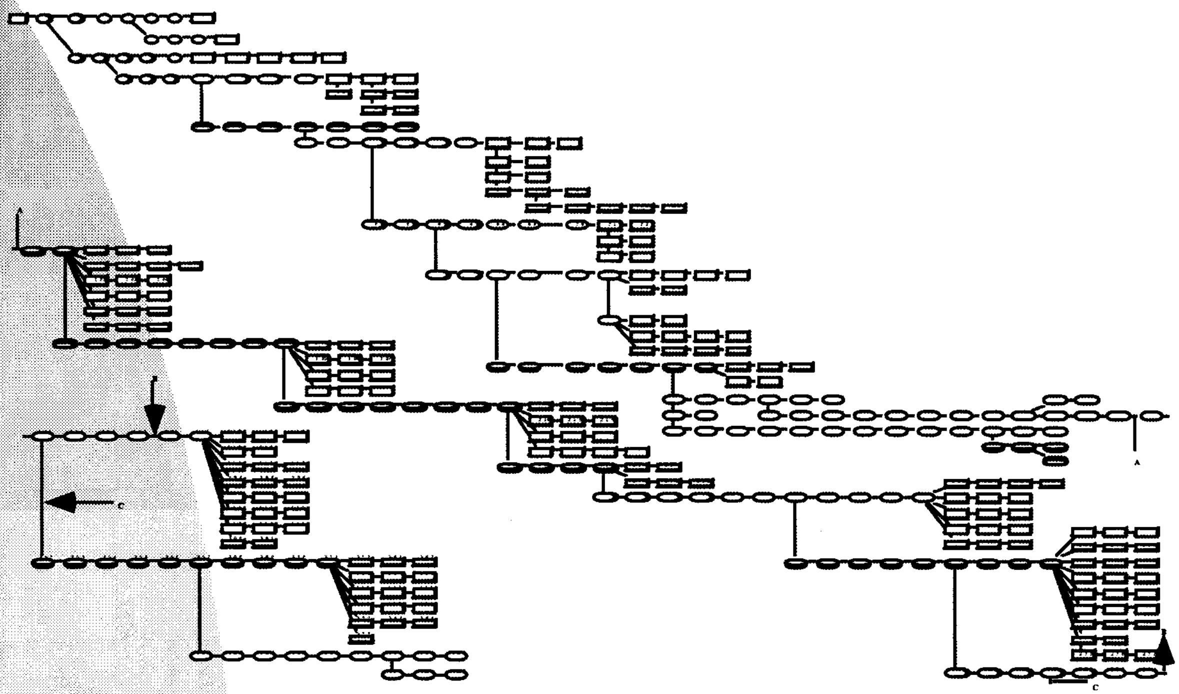
Measuring Structural Evolution



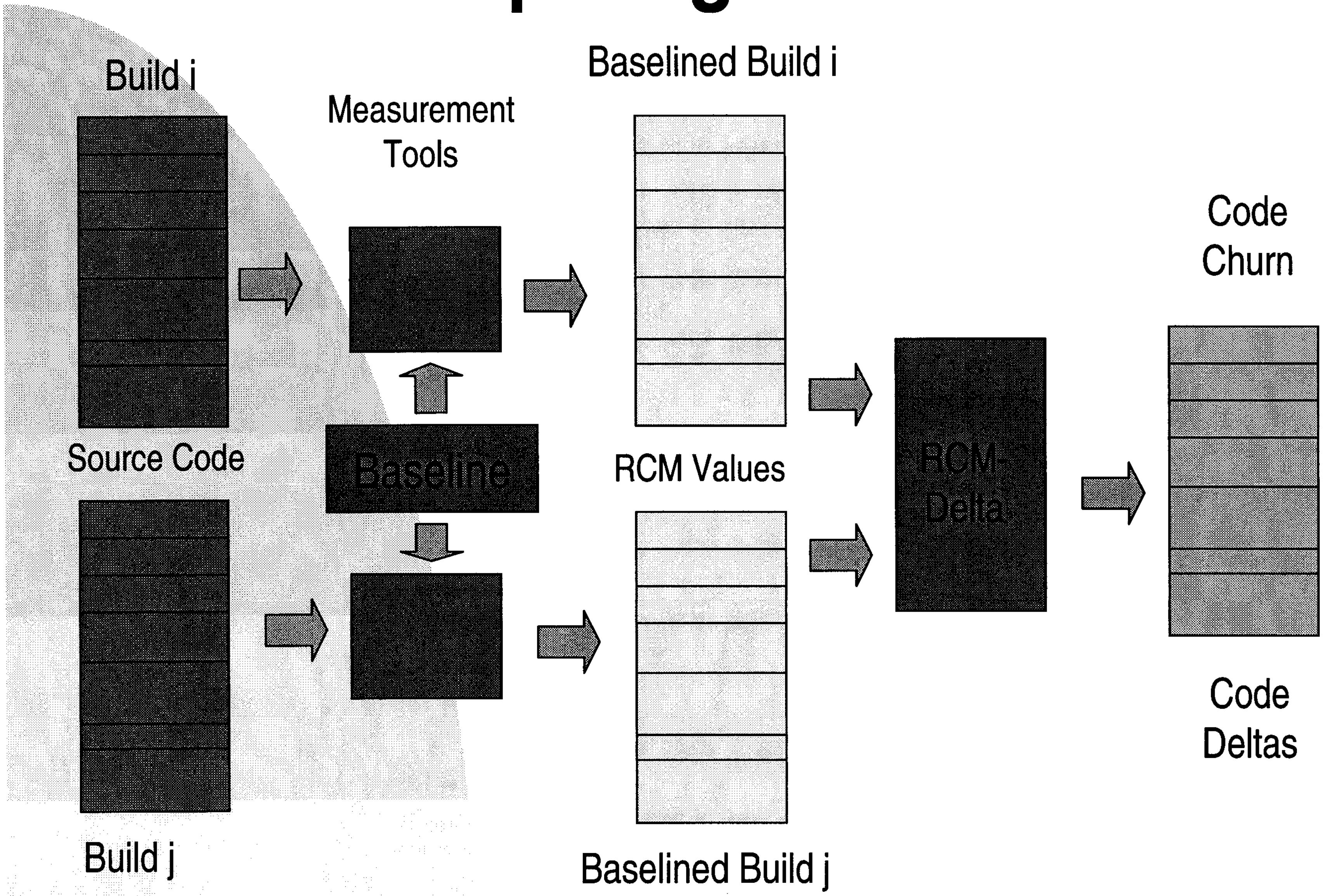
Relative Complexity

- Relative complexity is a synthesized metric
$$\rho_i^B = \sum_{j=1}^{m} \lambda_j^B d_j^B$$
- Relative complexity is a fault surrogate
  - Composed of metrics closely related to faults
  - Highly correlated with faults

Measuring Software Evolution



Comparing Two Builds



Measuring Evolution

- Different modules in different builds
  - $M_a^{i,j}$ set of modules not in latest build
  - $M_b^{i,j}$ set of modules not in early build
  - $M_c^{i,j}$ set of common modules
- Code delta $\delta_a^{i,j} = \rho_a^{B,j} - \rho_a^{B,i}$
- Code churn $\chi_a^{i,j} = \left|\delta_a^{i,j}\right| = \left|\rho_a^{B,j} - \rho_a^{B,i}\right|$
- Net code churn

$$\nabla^{i,j} = \sum_{m_c \in M_c} \chi_c^{i,j} + \sum_{m_a \in M_a^{i,j}} \rho_a^{B,i} + \sum_{m_b \in M_b^{i,j}} \rho_b^{B,j}$$

Estimating Fault Insertion Rate

- Proportionality constant, *k'*, representing the rate of fault insertion
- For j^{th} build, total faults inserted

$$F^{j} = kR^{0} + k'\Delta^{0,j}$$

- Estimate for the fault insertion rate

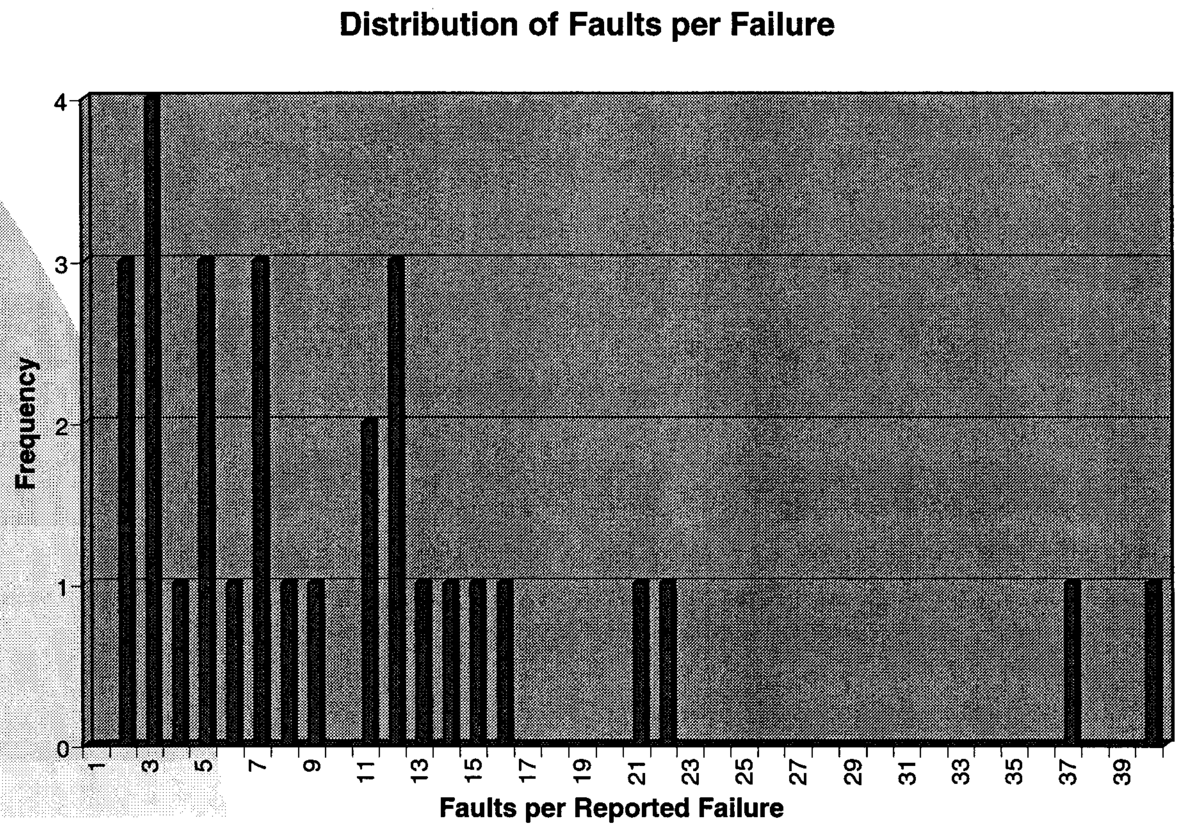
$$\begin{aligned} F^{j+1} - F^{j} &= kR^{0} + k'\nabla^{0,j+1} - kR^{0} + k'\nabla^{0,j} \\ &= k'(\nabla^{0,j+1} - \nabla^{0,j}) \\ &= k'\nabla^{j,j+1} \end{aligned}$$

Identifying and Counting Faults

- Unlike failures, faults are not directly observable
- fault counts should be at same level of granularity as software structure metrics
- Failure counts could be used as a surrogate for fault counts if:
  - Number of faults were related to number of failures
  - Distribution of number of faults per failure had low variance
  - The faults associated with a failure were confined to a single procedure/function

Actual situation shown on next slide

Observed Distribution of Faults per Failure



Statistics

	N		Mean	Median	Std. Deviation	Percentiles		
	Valid	Missing				25	50	75
Defects per 1 Failure	30	0	10.5667	7.5000	9.3428	3.7500	7.5000	13.2500

Fault Identification and Counting

- Faults must be identified at the module level
- To calibrate the regression model for fault insertion rates, for each fault repaired:
  - Determine the point at which it was first inserted into the module (e.g., inserted for version i of module A)
  - Compute the structural change between versions i and i-1 of module A

Fault Identification and Counting

- Rules have been developed to identify and count faults in source code.
- Tracing faults to their points of insertion becomes easier if there are links between the CM system and the problem reporting system (i.e., for a specific problem report, what source files were changed, and which versions of each source file repaired the fault?)
- Changes due to enhancements must be separated from changes due to fault repair

Estimating Test Efficiency

- Measures of structural evolution can be used together with profile information to estimate test efficiency.
  - Ideal profile - computed from cumulative structural change of modules since last test
  - Actual profile - observed during test execution
- Issue - instrumenting embedded real-time software system to obtain execution profile during test

Obtaining Execution Profile

- Build instrumentation into system
- Compile instrumentation into system
- Build execution profile logging capability into multi-mission simulator
  - For bit-level simulators being considered, appears to be specific instance of breakpoint capability
  - Behavior of software under test will not change (timing relationships will not be affected by instrumentation compiled into system)
  - Becomes part of institutional infrastructure, rather than being a project-to-project effort.